CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
COORDENAÇÃO DO CURSO TÉCNICO EM ELETRÔNICA

Felipe Bordoni Diniz

LAUNCHPAD F28377S

Volume 2: FPU, TMU, VCU, CLA

Belo Horizonte - MG
2018

Felipe Bordoni Diniz

LAUNCHPAD F28377S

Volume 2: FPU, TMU, VCU, CLA

Belo Horizonte - MG
2018

Resumo

O *DSP* TMS320F28377S produzido pela *Texas* Instruments™ é um poderoso dispositivo para controle de processos em malha fechada, devido a seus aceleradores de processamento que permitem efetuar operações em um número inferior de ciclos de *clock*, quando comparado com um microcontrolador.

Estes aceleradores estarem contidos num microcontrolador, fazem com que ele seja caracterizado como um *DSP*. O entendimento dos aceleradores de processamento é essencial para escolha de um dispositivo em função dos requisistos funcionais de uma aplicação.

Este documento apresenta informações acerca dos quatro aceleradores de processamento presentes no *DSP* TMS320F28377S, detalhando o funcionamento de cada um deles e como atuam para acelerar o processamento do dispositivo, apresentando testes realizados pela *Texas Instruments*™ que comprovam a eficácia destes aceleradores.

SUMÁRIO

Lista de Siglas.....6
Lista de Figuras.....8
Lista de Gráficos.....9
Lista de Quadros.....10
Lista de Tabelas.....11
1 Introdução.....12
***2* Características do *DSP* e *Launchpad*™**.....13
3 *Floating Point Unit* (*FPU*).....17
4 *Trigonometric Math Unit* (*TMU*).....21
5 *Viterbi, Complex Math, and CRC Unit* (*VCU*).....24
6 *Control Law Accelerator* (*CLA*).....27
7 Conclusões.....33
Referências Bibliográficas.....34

Lista de Siglas

ADC – Analog-Digital Conversor

AES – Advanced Encryption System

BRA – Branch

CLA – Control Law Accelerator

CRC – Cyclic Redundancy Check

D1 – Decode 1

D2 – Decode 2

DMA – Direct Memory Access

DSP – Digital Signal Processor

ePWM – Enhanced Pulse Width Modulation

EXE – Execute

F1 – Fetch 1

F2 – Fetch 2

FFT – Fast Fourier Transform

FIR – Finite Impulse Response

FPU – Floating Point Unit

GPIO – General Purpose Input/Output

IEEE – Institute of Electrical and Electronics Engineers

ISR – Interrupt Service Routines

LUF – Latched Underflow Flag

LVF – Latched Overflow Flag

MAC – Multiply and Accumulate

MOV – Move

NaN – Not a Number

PID – Proporcional Integral Derivativo

PIE – Peripheral Interrupt Expansion

PWM – Pulse Width Modulation

R1 – Read 1

R2 – Read 2

RB – Repeat Block Register

RPT – Repeat

RPTB – Repeat Block

STF – Floating-Point Status Register

TMU – Trigonometric Math Unit

VCU – Viterbi, Complex Math, and CRC Unit

W – Write

Lista de Figuras

Figura 2.1: Esquemático da *Launchpad™C2000™Delfino™* TMS320F28377S.....14
Figura 3.1: Normalização *IEEE-754* para ponto flutuante.....17
Figura 3.2: Diagrama em blocos da *FPU*.....19
Figura 5.1: Diagrama em blocos da *C28x+FPU+VCU*.....25
Figura 6.1: Diagrama de blocos da *CLA*.....28

Lista de Gráficos

Gráfico 2.1: Relação frequência x corrente.....16
Gráfico 2.2: Relação frequência x potência consumida.....16
Gráfico 5.1: Melhoras no processamento via *VCU*.....26

Lista de Quadros

Quadro 2.1: Consumo de corrente do dispositivo à 200 MHz.....15
Quadro 3.1: Comparação de processamento entre ponto flutuante e ponto fixo.....20
Quadro 4.1: Comparação de resultados entre *FPU* e *TMU*.....23
Quadro 6.1: Melhoras de desempenho devido à *CLA*.....32

Lista de Tabelas

Tabela 4.1: Instruções apresentadas pela *TMU*.....21
Tabela 4.4: Interpretações da representação em 32 *bits* em ponto flutuante.....22

1 Introdução

Em controle de processos digitais, a velocidade de aquisição de dados é uma característica vital para a qualidade dos resultados e da ação de controle do atuador no processo a ser controlado. Os microcontroladores em geral podem suprir a velocidade de processamento necessária para pequenas aplicações, em que o desempenho não precisa necessariamente ser tratado como algo vital, porém quando a velocidade de processamento é uma necessidade dominante, é necessário que sejam utilizados dispositivos que consigam atuar mais rapidamente.

Os *DSPs* (*Digital Signal Processors*) apresentam alguns aceleradores de processamento internos, que trabalham em conjunto com a *CPU* principal, permitindo que operações e aquisição de dados possam ser feitos com maior velocidade. Os *DSPs* possuem várias aplicações, que normalmente requerem alta velocidade de aquisição e processamento de dados, como por exemplo, controle de motores, reconhecimento de íris, processamento de áudio e vídeo, dentre outros.

Neste documento serão apresentados e explicados os aceleradores de processamento do *DSP* da série *C2000™Delfino™*TMS320F28377S, constituído pelo núcleo de processador C28x que trabalha na plataforma *Launchpad™* produzido pela *Texas Instruments™.* O *DSP* em questão possui quatro aceleradores de processamento, sendo eles a *FPU* (*Floating Point Unit*)*, TMU* (*Trigonometric Math Unit*)*, VCU* (*Viterbi, Complex Math, and CRC Unit*) e *CLA* (*Control Law Accelerator*).

2 Características do *DSP* e *Launchpad*™

Algumas das especificações do *DSP* TMS320F28377S são [1]:

- Arquitetura de 32 *bits*;
- Processador C281 (Contém os aceleradores *FPU, VCU* tipo 2*, TMU* tipo 0, *CLA* tipo 1);
- *Clock* de 200MHz;
- *ADC* de 16 *bits,* podendo trabalhar também como 12 *bits* (14 canais, podendo operar como *Differential* ou *Single Ended*);
- 15 canais *ePWM* do tipo 4;
- 3 *CPU Timers* de 32 *bits*;
- 41 pinos de *GPIO* (*General Purpose Input/Output*);

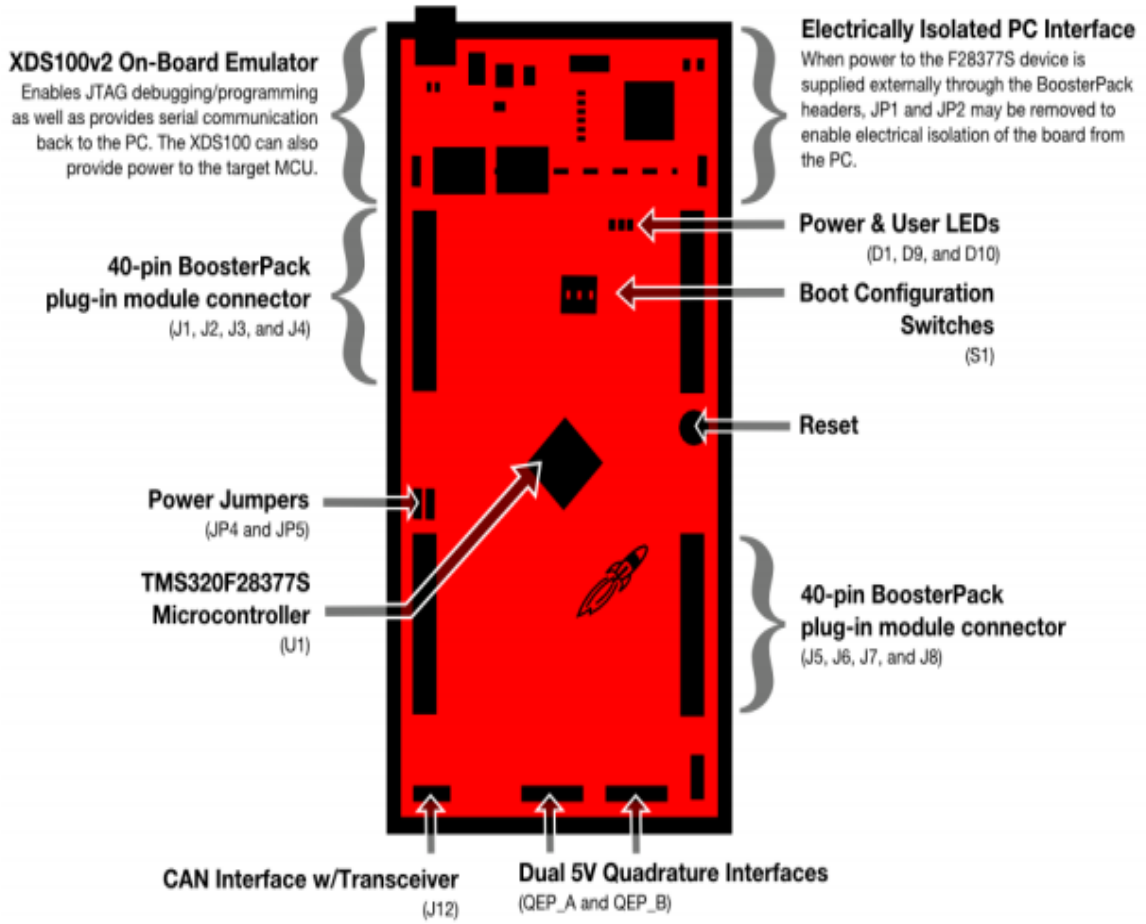
Nos aceleradores *VCU*, *TMU* e *CLA,* o "tipo" representa a adição de funções e características ao acelerador em questão, o que cada "tipo" significa na família *C2000*™ será explicado nos próximos capítulos deste documento.

A plataforma *Launchpad*™ já contém todos os hardwares necessários para que o F28377S possa operar. A ferramenta *JTAG* é a responsável pela aliança entre o *DSP* e o computador, onde o programador desenvolverá seu código de aplicação. A interface de comunicação serial UART é responsável pela comunicação através da conexão *USB*.

A *Launchpad*™ apresenta dois *LEDs* (acesos em nível lógico baixo) para aplicações do programador, uma chave *Push-Button* para *reset* e uma chave para configurações de *boot*. Na Figura 2.1, há uma ilustração esquemática que apresenta uma visão geral acerca destes recursos contidos na *Launchpad*™.

Figura 2.1: Esquemático da *Launchpad™C2000™Delfino™* TMS320F28377S.

Fonte: SPRUI25C. [2]



O Quadro 2.1, apresenta o consumo de corrente quando o *clock* de sistema do dispositivo está operando à 200 MHz. Para isto, o quadro compara diversos modos de operação do *DSP* com condições de testes específicas, e apresenta logo depois as correntes nominais para este teste, e as máximas permitidas, caso seja possível estabelecer um valor para a mesma. Estes valores podem flutuar e são susceptíveis a mudanças devido a multiplexação dos pinos utilizados na aplicação que envolva o dispositivo [1].

Quadro 2.1: Consumo de corrente do dispositivo à 200 MHz.

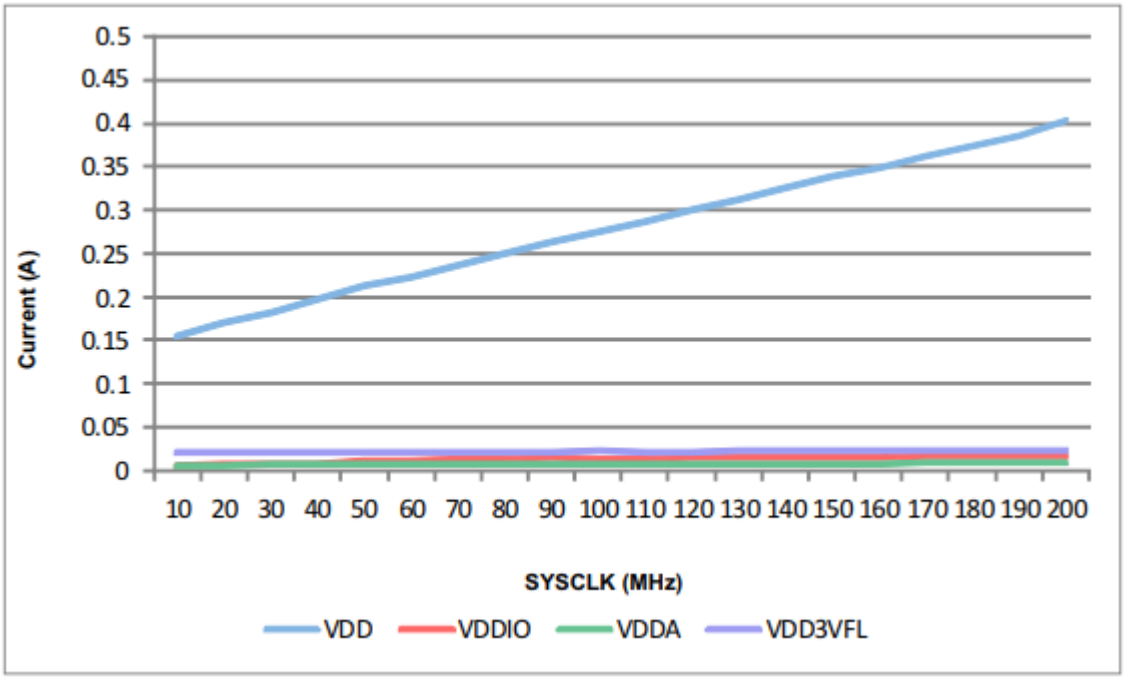
MODE	TEST CONDITIONS	I_{DD}		I_{DDIO} (1)		I_{DDA}		I_{DD3VFL}	
		TYP (2)	MAX (3)	TYP (2)	MAX (3)	TYP (2)	MAX (3)	TYP (2)	MAX (3)
Operational (RAM)	• Code is running out of RAM. (4) • All I/O pins are left unconnected. • Peripherals not active have their clocks disabled. • FLASH is read and in active state. • XCLKOUT is enabled at SYSCLK/4.	245 mA	360 mA	30 mA		13 mA	20 mA	33 mA	40 mA
IDLE	• CPU1 is in IDLE mode. • Flash is powered down. • XCLKOUT is turned off.	80 mA	185 mA	3 mA	10 mA	10 µA	150 µA	10 µA	150 µA
STANDBY	• CPU1 is in STANDBY mode. • Flash is powered down. • XCLKOUT is turned off.	30 mA	135 mA	3 mA	10 mA	5 µA	150 µA	10 µA	150 µA
HALT	• CPU1 watchdog is running. • Flash is powered down. • XCLKOUT is turned off.	1.5 mA	110 mA	750 µA	2 mA	5 µA	150 µA	10 µA	150 µA
HIBERNATE	• CPU1.M0 and CPU1.M1 RAMs are in low-power data retention mode.	300 µA	4 mA	750 µA	2 mA	5 µA	75 µA	1 µA	50 µA
Flash Erase/Program	• CPU1 is running from RAM. • All I/O pins are left unconnected. • Peripheral clocks are disabled. • CPU1 is performing Flash Erase and Programming. • XCLKOUT is turned off.	154 mA	230 mA	3 mA	10 mA	10 µA	150 µA	45 mA	55 mA

(1) I_{DDIO} current is dependent on the electrical loading on the I/O pins.
(2) TYP: V_{nom}, 30°C
(3) MAX: V_{max}, 125°C
(4) The following is executed in a loop on CPU1:
- All of the communication peripherals are exercised in loop-back mode: CAN-A to CAN-B; SPI-A to SPI-C; SCI-A to SCI-D; I2C-A to I2C-B; McBSP-A to McBSP-B; USB
- SDFM1 to SDFM4 active
- ePWM1 to ePWM12 generate 400-kHz PWM output on 24 pins
- CPU TIMERs active
- DMA does 32-bit burst transfers
- CLA1 does multiply-accumulate tasks
- All ADCs perform continuous conversion
- All DACs ramp voltage up/down at 150 kHz
- CMPSS1 to CMPSS8 active
- VCU does complex multiply/accumulate with parallel load
- TMU calculates a cosine
- FPU does multiply/accumulate with parallel load

Fonte: SPRS881D. [1]

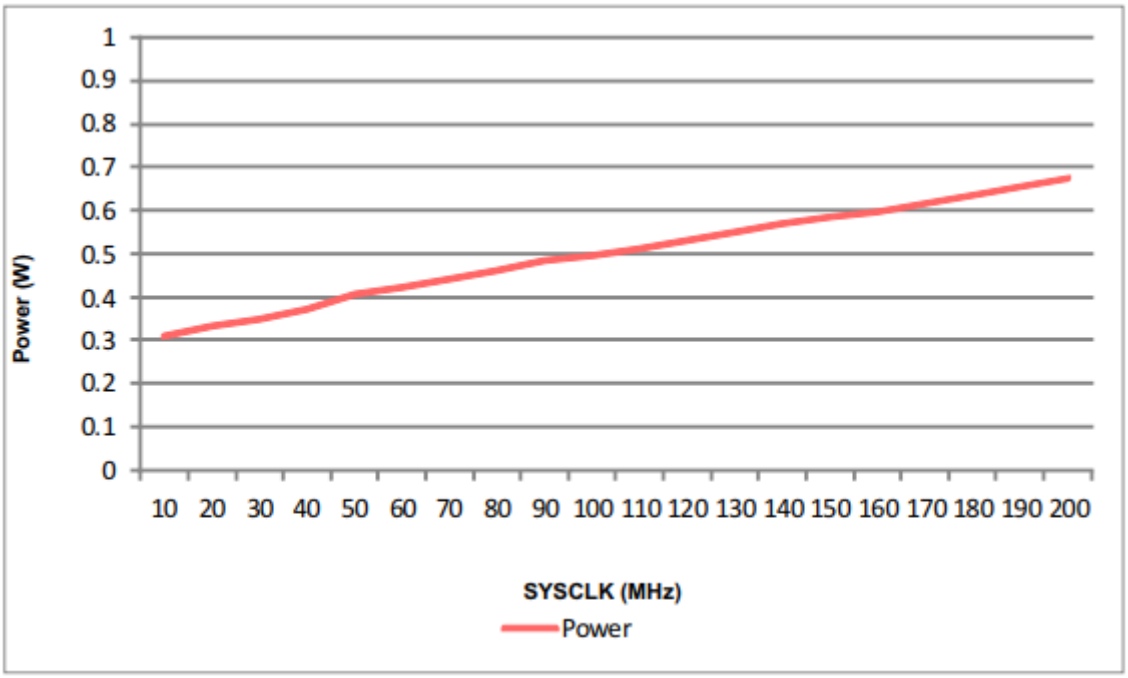
Os Gráficos 2.1 e 2.2 apresentam o consumo de corrente e a potência consumida pelo dispositivo de acordo com a frequência do *clock* do sistema. Observa-se que a relação frequência x corrente e frequência x potência é aproximadamente linear para os testes realizados pela *Texas Instruments*™. Assim como no Quadro 2.1, os resultados apresentados nos Gráficos 2.1 e 2.2 podem variar dependendo da aplicação a qual o *DSP* seja submetido [1].

Gráfico 2.1: Relação frequência x corrente.



Fonte: SPRS881D. [1]

Gráfico 2.2: Relação frequência x potência consumida.



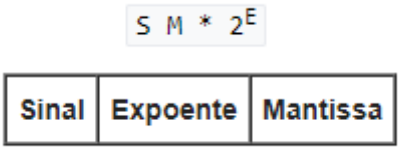
Fonte: SPRS881D. [1]

3 *Floating Point Unit* (*FPU*)

A *FPU* permite o *DSP* operar em potência de 10, facilitando computacionalmente as operações de multiplicação e divisão. A *IEEE* (*Institute of Electrical and Electronics Engineers*) estabeleceu o padrão *IEEE-754*, que dita os critérios para a representação de números binários em ponto flutuante.

A norma estabelecida pelo padrão *IEEE-754*, para operações em 32 *bits*, representa a divisão em três partes: o sinal, o expoente e a mantissa. Representado na Figura 3.1, onde S representa o sinal, M a mantissa, e E o expoente.

Figura 3.2: Normalização *IEEE-754* para ponto flutuante.



Fonte: Wikipedia. [3]

O ponto flutuante em 32 *bits* possui uma precisão de sete casas decimais, que é equivalente ao trabalhar com uma variável *float.* Também é comum referir a precisão desta representação apenas como precisão simples.

Na *IEEE-754* o primeiro *bit* é relacionado com o sinal, onde “0” significa um número positivo e “1” um número negativo. Os oito *bits* seguintes são referidos ao expoente, porém devem ser normalizados. A normalização é feita adicionando “127” ao expoente do número. Por fim, os 23 *bits* restantes representam a mantissa, que será preenchida pelos *bits* “à direita da vírgula” após a escrita em potência de 10_2.

Para exemplificar o processo, o número “-12.5” será convertido de decimal para binário em ponto flutuante. Como o número é negativo, o primeiro *bit* será 1. A conversão de “12.5” para binário resulta em 1100.1_2, que ao normalizar para a potência de 10_2, é expresso como $(1.1001_2) \times (10_2)^3$.

Com a conversão em potência de 10_2 finalizada, serão identificados os *bits* da mantissa e do expoente. A mantissa será representada pela sequência de *bits* à direita do ponto na representação em potência de 10_2. Logo a mantissa será representada por 1001_2. Os demais *bits* da mantissa serão preenchidos com “0”, então os *bits* da mantissa são $1001\,0000000000000000000_2$. Ao representarmos em potência de 10_2 o número “12.5”, o expoente encontrado foi “3”. A normalização do

expoente consiste em somá-lo à “127”, e converter o resultado para binário. Sendo assim o expoente normalizado é “130” na base decimal e 10000010_2 em binário. Ao final da conversão, o número “-12.5” será representado por:

$$1\,10000010\,1001\,0000000000000000000_2.$$

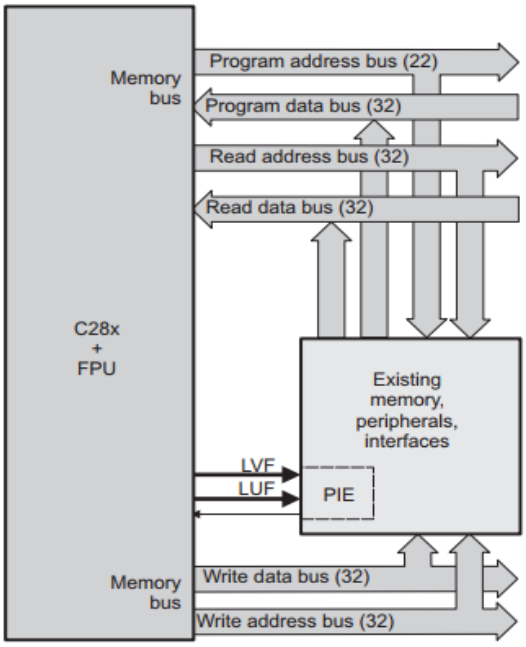
Com o auxílio do ponto flutuante, as operações com números muito grandes, ou muito pequenos, ficam fáceis de serem efetuadas. Devido a isto, surgiu a necessidade da criação de um acelerador de processamento específico, que ao trabalhar como uma extensão da CPU principal, permitiria que este método fosse factível.

No dispositivo TMS320F28377S, a *FPU* é munida por oito registradores de ponto flutuante *RnH* (*Floating-Point Result Register n*), onde *n* é o índice do registrador que varia de 0 a 7, um registrador de status, e um registrador de repetição em blocos, respectivamente denominados por *STF* (*Floating-Point Status Register*) e *RB* (*Repeat Block Register*). Todos estes registradores, exceto o *RB,* trabalham em modo *shadow*.

A implementação da *FPU* à *CPU* não ocasiona em mudanças nas instruções, arquitetura de barramentos de memória e na *pipeline* originais da *CPU*. A Figura 3.2 ilustra a implementação da *FPU* com o processador C28x e seus periféricos.

A arquitetura do *DSP* na Figura 3.2 não perdeu sua identidade, apenas foi acrescida de novas ligações entre a *PIE* e a *FPU*, que agora está trabalhando integrada à *CPU* principal.

Figura 3.3: Diagrama em blocos da *FPU*.



Fonte: SPRUHS1A. [4]

O *STF* reflete os resultados das operações em ponto flutuante realizadas pelo dispositivo, e possui três regras básicas para os flags destas operações [4]:

1. Flags zero e negativo são setados baseados em operações de *MOV* (*Move*) entre registradores;
2. Flags zero e negativo são setados baseados em resultados de operações de comparações, mínimo, máximo, negativo e valor absoluto;
3. Flags *overflow* e *underflow* são setados por instruções matemáticas como multiplicações, adições, subtrações e operações de inversão $\left(\frac{1}{x}\right)$. Estes flags também podem estar conectadas ao bloco de *PIE* (*Peripheral Interrupt Expansion*) do dispositivo.

O *RB* ao ser incorporado com a *CPU* cria uma nova instrução, chamada de *Repeat block instruction*, e permite a repetição de uma parte do código utilizando poucos ciclos de *clock*. No exemplo de conversão do número “-12.5” de decimal para binário em ponto flutuante, após a escrita dos quatro primeiros *bits* da mantissa, os outros 19 *bits* são zeros. A função *RPTB* (*Repeat Block*) faz com que este processo de escrita ocorra mais rapidamente do que se feito apenas pela *CPU* principal. A função possui dois argumentos, o primeiro é o *label* que indica para a

função até onde devem-se repetir as instruções. O segundo argumento é número de repetições a serem executadas. Este número pode ser um valor fixo no código, ou lido de um registrador. A sintaxe desta instrução em *Assembly* é dada a seguir [5].

; Repeat Block com contagem direta:

***RPTB** #label, #RC ; 1 ciclo*

... ; outras instruções que demandam n ciclos

label

; label: label após a última instrução que fará parte da repetição

; RC: número de repetições desejadas

;

; Repeat Block com contagem armazenada em um registrador

***RPTB** #label, loc16 ; 4 ciclos*

... ; outras instruções que demandam n ciclos

label

; label: label após a última instrução que fará parte da repetição

; loc16: registrador que contém o valor de repetições desejadas

Apenas a primeira iteração de *RPTB* demanda ciclos de *clock,* as outras iterações não necessitam de ciclos de *clock* para que sejam executadas, isto é o grande trunfo desta função. Caso opte-se por utilizar a função *RPTB* com contagem direta, são gastos (1+n) ciclos de *clock*, caso seja utilizado um registrador para armazenar o número de repetições, a função demanda (4+n) ciclos de *clock*.

De acordo com experimentos feitos pela *Texas Instruments*™, a *FPU* apresenta melhoras na velocidade de processamento em comparação às operações em ponto fixo. O Quadro 3.1 realiza uma comparação entre estas duas formas de

Quadro 3.2: Comparação de processamento entre ponto flutuante e ponto fixo.

Function	Type	FPU Cycles	Fixed Cycles	Comments Improvements/Comments
Complex FFT	512 pt	24243	63192	2.61x (FPU vs Fixed Point)
	1024 pt	53219	141037	2.65x (FPU vs Fixed Point)
Real FFT	512 pt	13670	34513	2.52x (FPU vs Fixed Point)
	1024 pt	30352	76262	2.51x (FPU vs Fixed Point)
Square Root	Compiler intrinsic	22	64	2.91x (FPU vs Fixed Point)
Finite impulse response (FIR)	64 pts	119	111	1.07 (Fixed vs FPU) - Both algorithms make use of the circular addressing mode of the C28x

Fonte: SPRY288A [6]

Nos dois primeiros casos do Quadro 3.1, tanto a *FFT* (*Fast Fourier Transform*) complexa quanto a real tiveram uma performance melhor quando tratadas com a

FPU implementada, alcançando em todos os experimentos reduções de ciclos gastos superiores à 2.5 vezes. No terceiro caso, tratou-se da operação de raiz quadrada, que apresentou uma redução de ciclos de 2.91 vezes. No último caso, o *FIR* (*Finite Impulse Response*), a *FPU* não apresentou melhoras no processamento, e a operação em ponto fixo apresentou uma velocidade 1.07 vezes maior que a da *FPU*.

Com os testes realizados pela *Texas Instruments*™ pode-se concluir que a *FPU* trouxe efetivos benefícios ao processamento da *CPU*, proporcionando que as *FFTs* e raízes quadradas fossem realizadas em menos ciclos. No caso do *FIR*, como tanto o ponto flutuante quanto o ponto fixo utilizam o mesmo tipo de endereçamento da *CPU* principal, é de se esperar que a *FPU* demande mais ciclos devido ao algoritmo implementado.

4 *Trigonometric Math Unit* (*TMU*)

A *TMU* melhora a performance na execução do conjunto de instruções trigonométricas e aritiméticas realizadas pela *FPU-C28x.* No TMS320F28377S, a *TMU* é do tipo 0, que caracteriza o módulo padrão [7].

Com a implementação da *TMU*, são integradas novas funções ao conjunto de instruções já existente. A priorização da utilização das novas funções é justificada pelo foco da *TMU* em computar com maior velocidade estas operações. A Tabela 4.1 apresenta as funções interpretadas pela *TMU*.

Tabela 4.1: Instruções apresentadas pela *TMU*.

Fonte: SPRUHS1A. [4]

Instructions	C Equivalent Operation
MPY2PIF32 RaH,RbH	a = b * 2pi
DIV2PIF32 RaH,RbH	a = b / 2pi
DIVF32 RaH,RbH,RcH	a = b/c
SQRTF32 RaH,RbH	a = sqrt(b)
SINPUF32 RaH,RbH	a = sin(b*2pi)
COSPUF32 RaH,RbH	a = cos(b*2pi)
ATANPUF32 RaH,RbH	a = atan(b)/2pi
QUADF32 RaH,RbH,RcH,RdH	Operation to assist in calculating ATANPU2

As instruções existentes antes da introdução da *TMU* não foram afetadas, assim como a arquitetura de barramento de memória e *pipeline*. Todas as instruções executadas pela *TMU* utilizam o conjunto de registradores da *FPU* (*R0H* até *R7H*) para cumprir suas operações.

Segundo o padrão *IEEE-754,* os valores apresentados nos *bits* de sinal, expoente e mantissa, podem remeter a resultados que devem ser interpretados de maneiras especiais. A Tabela 4.2 apresenta todos os tipos de interpretações que podem ser tidas ao longo de todas as representações possíveis em 32 *bits*.

Tabela 4.2: Interpretações da representação em 32 *bits* em ponto flutuante.

S32	E32 (7:0)	M32 (22:0)	Value (V)
0	0	0	Zero (V = 0)
1	0	0	Negative Zero (V = -0)
0 +ve 1 -ve	0	non zero	De-normalized ($V=(-1)^{S*} 2^{(-126)*} (0.M)$)
0 +ve 1 -ve	1 to 254	0 to 0x7FFFFF	Normal Range ($V=(-1)^{S} * 2^{(E-127)} * (1.M)$)
0	254	0x7FFFFF	Positive Max (V = +Max)
1	254	0x7FFFFF	Negative Max (V = -Max)
0	max=255	0	Positive Infinity (V = +Infinity)
1	max=255	0	Negative Infinity (V = -Infinity)
x	max=255	non zero	Not A Number (V = NaN)

Fonte: SPRUHS1A. [4]

O dispositivo pode encontrar o valor de zero com sinal positivo ou negativo, porém a *TMU* sempre interpreta os “zeros negativos” como “zeros positivos”. Em todas as operações da *TMU*, são gerados “zeros positivos”, onde o *bit* de sinal é zero (S=0), todos os *bits* do expoente são zero (E=0) e todos os *bits* da mantissa também são zero (M=0).

Os números não normalizados são aqueles em que todos os *bits* do expoente são zero (E=0), porém o número representado na mantissa é diferente de zero (M≠0). O valor do *bit* de sinal é irrelevante para esta interpretação. Estes números são interpretados pela *TMU* como zeros, fazendo com que o valor dos *bits* da

mantissa seja levado à zero (M=0). As operações da *TMU* nunca geram valores não normalizados.

O *underflow* ocorre quando uma operação gera um valor, em módulo, muito pequeno para ser representado no formato de ponto flutuante. Nestes casos, o valor retornado é zero. Caso uma operação da *TMU* ocasione em um *underflow*, então o *flag* de *underflow LUF* (*Latched Underflow Flag*) é setado para o nível lógico alto, este estado lógico permanece até que uma operação o limpe.

Análogo ao *underflow* existe o *overflow* que possui o mesmo princípio, porém trata de números muito grandes, em módulo, para serem expressados no formato de ponto flutuante. Em casos de *overflow*, o valor retornado é de ± infinito. Caso uma operação da *TMU* retorne um *overflow*, será setado em nível lógico alto o flag de *overflow LVF* (*Latched Overflow Flag*), este estado lógico permanece até que uma operação o limpe.

Em casos em que o expoente é máximo (E=Máximo) e os *bits* da mantissa são diferentes de zero (M≠0), o valor é considerado como um *NaN* (*Not a Number*), porém é tratado como infinito, com o expoente máximo (E=Máximo), e os *bits* da mantissa como zero (M=0) para todas as operações. As operações da *TMU* nunca geram um valor *NaN,* ao invés disto, geram o valor infinito.

Na maioria das aplicações em controle de processos que envolvam *DSPs*, eles possuem a vital necessidade de uma *TMU* para viabilizar, por exemplo, o cálculo de transformadas matemáticas. Em máquinas trifásicas síncronas, as transformadas de *Clarke* e *Park* são fundamentais para o processo de análise e controle. [8]

Devido as operações que envolvem diversas técnicas de álgebra linear, operações trigonométricas e números complexos [9] (que serão de responsabilidade da *VCU,* que será detalhada no próximo capítulo), a necessidade da *TMU* é vital para que estes cálculos sejam realizados em tempo hábil.

A *Texas Instruments*™ realizou testes com as Transformadas de *Park* para comprovar a eficácia da *TMU*. A *FPU* demanda tipicamente entre 80 e 100 ciclos para executar uma Transformada de *Park*, enquanto que com a implementação da *TMU* este número cai para 13 ciclos, implicando em uma melhora de 85% no processamento [6].

Outros experimentos foram feitos pela *Texas Instruments*™ envolvendo o controle de motores e placas solares para realizar a comparação entre dispositivos munidos de *FPU* e de *FPU+TMU*. Os resultados destes experimentos estão exibidos na Tabela 4.3 e detalhados em seguida.

Quadro 4.3: Comparação de resultados entre *FPU* e *TMU*.

Application	Number of Execution Cycles		Improvement
	FPU	TMU	
	Min/Max	Min/Max	
Motor AC Induction	888/952	593/670	1.42x (vs FPU)
Motor Permanent Magnet	783/786	547/592	1.32x (vs FPU)
Solar 3-Phase	1351/1358	985/983	1.38x (vs FPU)

Fonte: SPRY288A. [6]

No primeiro experimento, foi utilizado um motor de indução de corrente alternada, que com a implementação da *TMU* em conjunto à *CPU* e *FPU*, apresentou uma melhora no número de ciclos de 1.42 vezes em relação à mesma aplicação, porém sem a utilização da *TMU*. No segundo experimento, foi utilizado um motor de imã permanente, que novamente apresentou redução no número de ciclos utilizando a *TMU*. A melhora no processamento foi de 1.32 vezes em relação à mesma experiência feita com a utilização da *FPU*. Por fim, o último experimento utilizou uma placa solar trifásica, e mais uma vez a *TMU* mostrou diminuição no número de ciclos para a tarefa, desta vez a melhora foi de 1.38 vezes em relação aos ciclos demandados pela *FPU*.

Em todos os casos a *TMU* foi superior à *FPU*, mostrando assim que a implementação da *TMU* à *CPU+FPU* acarreta em uma significante melhora na performance do *DSP*.

5 *Viterbi, Complex Math, and CRC Unit* (*VCU*)

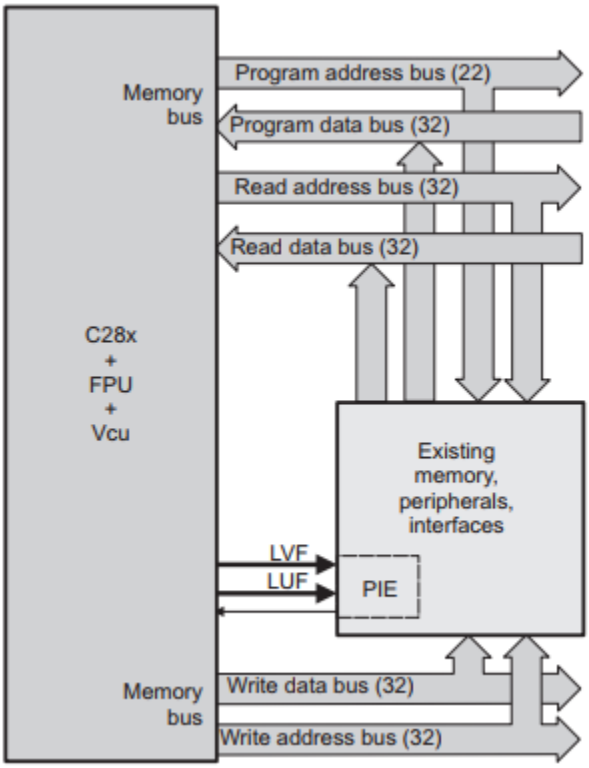
A *VCU* acelera a performance dos algoritmos baseados em comunicação por um fator de sete vezes [6]. No *DSP* TMS320F28377S, a *VCU* é do tipo 2, que possui incrementos em relação aos tipos 0 e 1 (que são equivalentes), e representam a *VCU* mais básica da *Texas Instruments*™. A *VCU* tipo 2, ou *VCU-II,* possui a adição de instruções para melhorar a performance na decodificação *Viterbi,* instruções aritiméticas complexas, *FFT* complexa, encriptação e desencriptação *AES*

(*Advanced Encryption System*). As instruções adicionadas são relacionadas à *DMA* (*Direct Memory Access*) entre a *VCU* e o processador *C28x*. Também foi adicionada a interrupção da *VCU* à *PIE*.

A decodificação *Viterbi* é comumente utilizada em aplicações de comunicações de banda base. O algoritmo de decodificação *Viterbi* consiste em três partes principais, que são os cálculos métricos de ramo, seleção de comparação (*Viterbi Butterfly*), e operação de rastreamento. Os algoritmos da *CRC* (*Cyclic Redundancy Check*) oferecem um método direto para verificação da integridade de dados em grandes blocos de memória, pacotes de comunicação ou seções de código. A *VCU* também interpreta números complexos com a seção de *Complex Math*, que possui aplicações em *FFTs*, espectros via *FFTs* complexas, processamento de sinais, filtros complexos, dentre outras [4].

A *VCU* amplia a capacidade da *CPU* e *FPU* por acrescentar instruções, e assim como nos casos da *TMU* e *FPU*, não são feitas alterações nas instruções já existentes, *pipeline* e arquitetura de barramentos de memória. Como ocorre apenas a adição de instruções, o diagrama em blocos da *C28x+FPU+VCU* é praticamente igual ao diagrama apresentado na Figura 2.1, como mostrado na Figura 5.1. A mesma não mostra, porém há a implementação da interrupção da *VCU* à *PIE*, como citado anteriormente.

Figura 5.4: Diagrama em blocos da *C28x+FPU+VCU*.



Fonte: SPRUHS1A. [4]

A *VCU* é munida de um conjunto de nove registradores de resultados (*VR0* a *VR8*), dois registradores de *Traceback* (*VT0* e *VT1*), um registrador de configuração e *status* (*VSTATUS*), um registrador de resultados da *CRC* (*VCRC*) e um registrador de bloco de repetição (*RB*).

Além de outras diversas funções que a *VCU* incorpora ao dispositivo [4], uma das principais está referida à Unidade de Aritmética Matemática Complexa, permitindo somar, subtrair e multiplicar números complexos. A *VCU* também implementa a função *MAC* (*Multiply and Accumulate*), permitindo que as multiplicações recorrentes sejam feitas rapidamente. A utilização conjunta da função *RPT* (*Repeat*) da linguagem *Assembly* com a função *MAC* (*RPT* || *MAC*) torna as operações em potência computacionalmente viáveis.

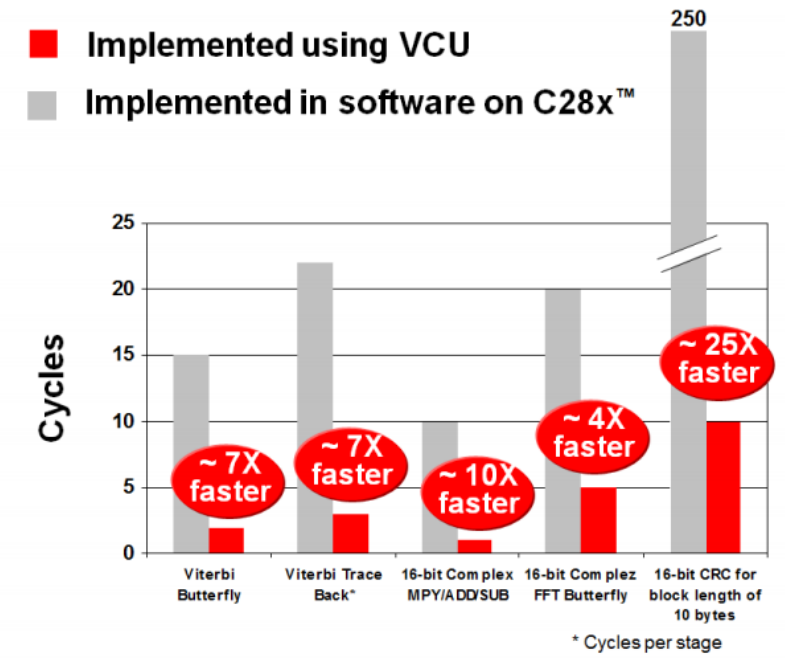
Operações de *BRA* (*Branch*) em *Assembly* com o auxílio da *VCU* são realizadas em apenas um ciclo de *Clock,* o que para o tratamento de interrupções é uma característica desejável. A *Viterbi Butterfly*, sem o auxílio da *VCU*, demanda quinze ciclos de *clock* por estágio para ser efetuada, enquanto que com a *VCU,* a *Viterbi Butterfly* demanda apenas dois ciclos por estágio. A operação de *Traceback*

sem a *VCU* demanda 22 ciclos de *clock*, enquanto que com a *VCU* a operação necessita apenas de três ciclos por estágio [6].

Em aplicações de comunicações e processamento de sinais, a *FFT* e gráficos de espectros são necessidades vitais no processo. A *VCU* para *FFTs* de 16 *bits* em ponto fixo complexas (magnitude e fase) demandam 5 ciclos de *clock* por estágio para serem efetuadas. Sem a utilização da *VCU*, o processo demandaria 20 ciclos por estágio [6].

Estas informações estão graficamente mostradas no Gráfico 5.1, proporcionando a comparação em número de ciclos entre a implementação utilizando a *VCU* e a utilização apenas do *C28x*™.

Gráfico 5.3: Melhoras no processamento via *VCU*.



Fonte: SPRY288A. [6]

6 *Control Law Accelerator* (*CLA*)

A *CLA* é um processador independente em ponto flutuante com 32 *bits* utilizado para acelerar a obtenção de leituras do *ADC* e proporcionar a redistribuição do processamento entre a *CPU* principal e a *CLA.* Com a possibilidade da *CLA* acessar diretamente as conversões realizadas pelo *ADC*, as leituras são realizadas *just-in-time*. Sendo assim, a *CPU* principal fica livre para executar outras tarefas do sistema, aliviando a sobrecarga de processamento.

A *CLA* encontrada no TMS320F28377S é do tipo 1, que possui melhoras com relação ao módulo padrão tipo 0, como por exemplo, a possibilidade de uma tarefa disparar uma interrupção para a *CPU* principal. A Figura 6.1 ilustra o diagrama de blocos da *CLA* tipo 1.

As principais características da *CLA* são [10]:

- Opera na mesma frequência de *clock* da *CPU* principal;
- Arquitetura independente permitindo execução de algoritmo independente da *CPU* principal;
- Conjunto de instruções;
- O código da *CLA* pode conter até oito tarefas ou rotinas de interrupção.

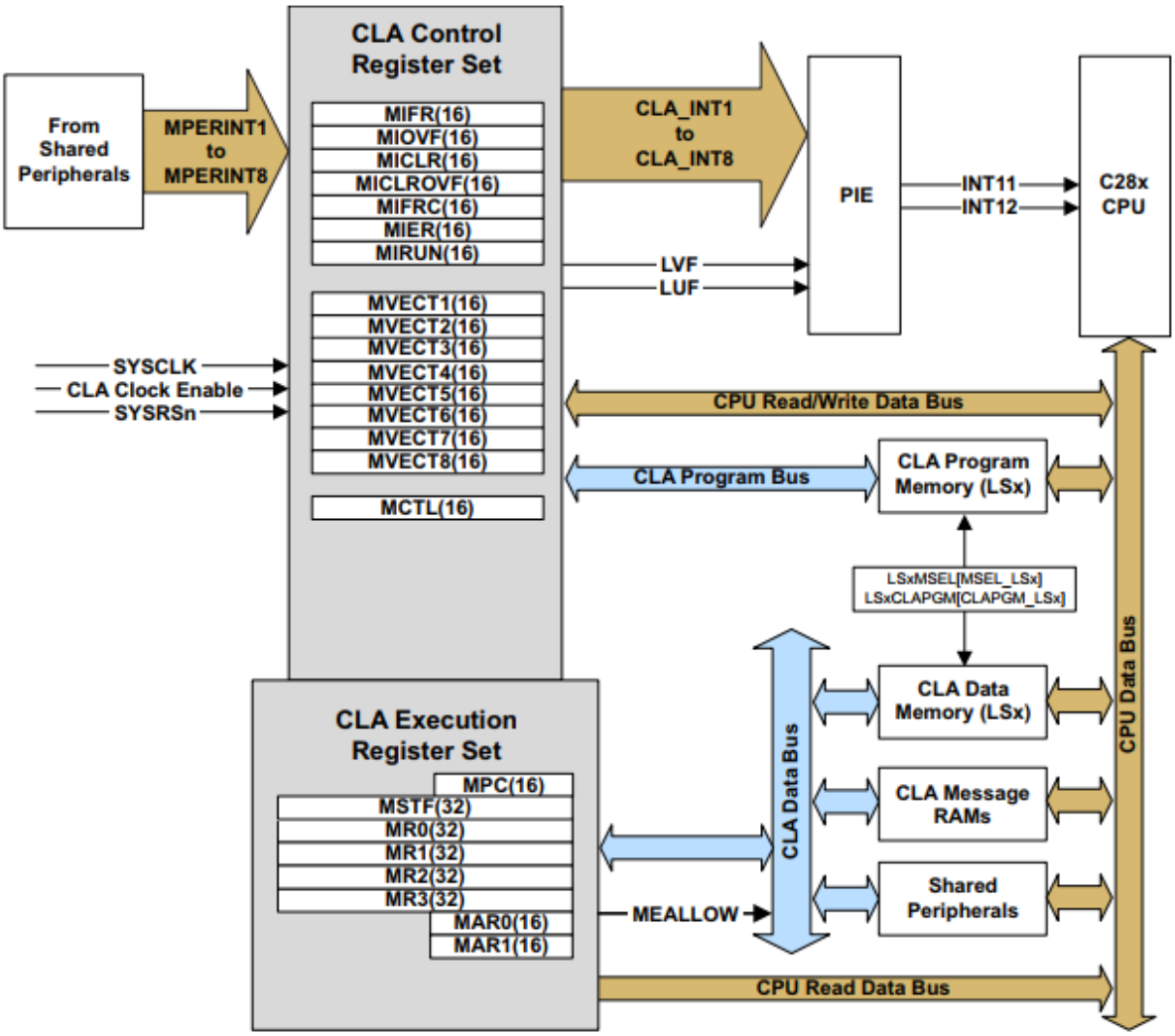
A *CLA* pode acessar três tipos de memória:

1. Memória de programa;
2. Memória de dados;
3. *RAMs* mensageiras.

O código de programação da *CLA* pode ser gravado em qualquer uma das memórias locais compartilhadas (LSxRAM). Em *reset,* todos os blocos da memória são mapeados para a *CPU* principal [5].

Qualquer uma das LSxRAMs do dispositivo pode operar como um bloco de memória de dados para a *CLA.* Em *reset*, todos os blocos são mapeados para o espaço de memória da *CPU* principal, podendo fazer com que sejam gravados nestes blocos dados, tabelas, coeficientes, para uso da *CLA*.

Figura 6.5: Diagrama de blocos da *CLA*.



Fonte: SPRUHX5E. [10]

As *RAMs* mensageiras compartilhadas entre a *CLA* e *CPU* são usadas para compartilhar dados e estabelecer a comunicação entre os dois processadores. Existem dois pequenos blocos de memória para estas comunicações, as *RAMs* mensageiras são posicionadas entre a *CLA* e a *CPU* principal em cada subsistema da *CPU*. Estas *RAMs* permitem ao usuário apenas a leitura. Para a troca de informações entre processadores, as *RAMs* tornam-se suceptíveis a escrita apenas da *CLA* ou *CPU*, dependendo do fluxo da transmissão:

- Transmissão de informação *CLA* → *CPU*:

  A *CLA* utiliza estre bloco de memória para transmitir a informação para a *CPU* principal. Sendo assim, o bloco pode ser escrito e lido pela *CLA*, porém apenas lido pela *CPU* principal. As operações de escrita feitas pela *CPU* principal são ignoradas.

- Transmissão de informação *CPU* → *CLA*:
  A *CPU* utiliza estre bloco de memória para transmitir a informação para a *CLA*. Sendo assim, o bloco pode ser escrito e lido pela *CPU* principal, porém apenas lido pela *CLA*. As operações de escrita realizadas pela *CLA* são ignoradas.

A *CLA* possui três barramentos, o de programa, o de leitura de dados e o de escrita de dados. Esta arquitetura de barramentos é totalmente dedicada à *CLA,* e similar à arquitetura encontrada na *CPU* principal. Sendo assim num único ciclo de clock, a *CLA* realiza operações de leitura, escrita e executa instruções.

O barramento de programa tem um alcance de acesso de 32k instruções de 32 *bits*. Como todas as instruções da *CLA* são de 32 *bits*, o alinhamento entre espaços de memória sempre ficará alinhado em um endereço par. A quantidade de espaço de programa disponível para a *CLA* é limitado ao número de blocos LSxRAM disponíveis.

Os barramentos de escrita e leitura possuem um alcance de endereçamento de 64k x 16, e podem executar leituras e escritas de 16 ou 32 *bits*. Estes barramentos possuem acesso à memória de dados da *CLA,* aos periféricos compartilhados, e em relação às *RAMs* mensageiras, apenas o barramento de leitura de dados possui acesso à ambos os blocos mensageiros, enquanto o barramento de escrita de dados só possui acesso a *RAM* mensageira quando o fluxo da transmissão é da *CLA* para a *CPU*.

O código de programa da *CLA* é dividido entre tarefas e *ISR* (*Interrupt Service Routines*), as tarefas não possuem uma posição fixada de início ou tamanho. A *CLA* identifica o endereço de início de uma tarefa a partir do conteúdo do vetor de interrupção associado (*MVECT1* a *MVECT8*), e o final é indicado pela instrução *MSTOP.* A *CLA* suporta até oito tarefas, sendo a tarefa 1 a de maior prioridade, enquanto a tarefa 8 possui a menor prioridade. Estas tarefas podem ser requisitadas por uma interrupção de um periférico ou por software.

Cada uma das tarefas da *CLA* possui 256 fontes de interrupções, e o *trigger* associado a cada uma dessas tarefas é definido pela escrita nos *bits* do registrador *DmaClaSrcSelRegs.CLA1TASKSRCSELx[TASKz]*, onde ‘z’ é o número da tarefa a ser associada a um *trigger* e ‘x’ é o registrador que contém a tarefa em específico, para *TASK1* a *TASK4* deve-se selecionar *CLA1TASKSRCSEL1*, e para *TASK5* a

TASK8 deve-se selecionar *CLA1TASKSRCSEL2*. Cada um dos registros de *TASKz* possui 8 *bits,* e cada sequência destes *bits* é relacionada a uma fonte diferente de *trigger.*

As tarefas também podem ser inicializadas via software da *CPU* principal pela escrita ao registrador *MIFRC* ou pela instrução *IACK.* A utilização da instrução *IACK* possui maior eficiência devido a não necessitar das instruções *EALLOW* e *EDIS*, que são necessárias para a escrita em *MIFRC* devido a proteção do registro. Para a escrita em *IACK,* basta realizar a escrita em *MCTL[IACKE]* setando o *bit* em nível lógico alto, e após isto, realizar a escrita nos 8 *bits* de *IACK,* setando em nível lógico alto os *bits* correspondentes às tarefas que o programador deseja trabalhar com a interrupção via software. Por exemplo, caso o programador deseje trabalhar com as tarefas 1, 3 e 7, ele deve escrever a sequência 01000101_2 nos *bits* de *IACK*. A programação da *CLA* pode ser feita em linguagem C (com certas restrições), ou via *Assembly*.

A *pipeline* da *CLA* é bem similar à *pipeline* da *CPU* principal, possuindo oito estágios [10]:

1. *F1* (*Fetch 1*):
   Durante o estágio *F1*, o endereço de leitura do programa é colocado no barramento de programa da *CLA*;
2. *F2* (*Fetch 2*):
   Durante o estágio *F2*, a instrução é lida usando o barramento de leitura de dados da *CLA*;
3. *D1* (*Decode 1*):
   Durante o estágio *D1*, a instrução é decodificada;
4. *D2* (*Decode 2*):
   Gera o endereço de leitura de dados. *Branches* condicionais são realizados baseados nos *flags* do registrador *MSTF*;
5. *R1* (*Read 1*):
   Coloca o endereço de leitura de dados no barramento de leitura de dados da *CLA*;
6. *R2* (*Read 2*):
   Lê o dado usando o barramento de leitura de dados da *CLA*;
7. *EXE* (*Execute*):

Executa a operação;

8. *W* (*Write*):
   Coloca o endereço de escrita, e o dado a ser escrito no barramento de escrita de dados da *CLA*.

O *reset* da *CLA* pode ser necessário em certos momentos, como por exemplo, caso a mesma entre em um *loop* infinito. Existem dois tipos de *resets* para a *CLA*, o *hard reset* e o *soft reset.* Ambos podem ser executados pelo *debugger* ou pela *CPU* principal.

- *Hard reset*:
  A escrita do nível lógico alto ao *bit* de *HARDRESET* no registrador *MCTL* irá causar um *hard reset* à *CLA*. O comportamento deste *reset* é o mesmo de um *reset* do sistema, todas as configurações da *CLA* e registros de execução serão setados para seu estado *default* e a execução da *CLA* irá parar.
- *Soft reset*:
  A escrita do nível lógico alto ao *bit* de *SOFTRESET* no registrador *MCTL* irá causar um *soft reset* à *CLA*. Caso uma tarefa esteja sendo executada, ela irá parar e os *bits* do *MIRUN* associado serão limpos (*MIRUN* contém os *bits* da tarefa em execução). Todos os *bits* dentro do registro de habilitação de interrupção (*MIER*) também serão limpos, sendo assim nenhuma nova tarefa será inicializada.

Os testes feitos pela *Texas Instruments*™ acerca da eficácia da implementação da *CLA* em relação à *CPU* principal trabalhando sozinha foram realizados em tarefas que demandavam controles de motores de corrente alternada de indução, e controles de potência em placas solares via PIDs de 2p2z (2 pólos e 2 zeros) e 3p3z (3 pólos e 3 zeros) [6]. Os resultados estão apresentados no Quadro 6.1, e discutidos após o mesmo:

Quadro 6.4: Melhoras de desempenho devido à *CLA*.

Application	Number of Execution Cycles		Improvement
	CPU	CLA	
	Min/Max	Min/Max	
Motor AC Induction	888/952	639/694	1.39x (vs CPU)
Power CNTL 2p2z	48	39	1.23x (vs CPU)
Power CNTL 3p3z	68	52	1.31x (vs CPU)

Fonte: SPRY288A. [6]

Na primeira aplicação, foi utilizado um motor de indução de corrente alternada, e ao se comparar a performance da *CLA* e *CPU*, foi verificado que a *CLA* obteve um número de ciclos 1.39 vezes menor que o da *CPU*. Na segunda e terceira aplicação, foram tratadas funções de transferência de 2 pólos e 2 zeros, e 3 pólos e 3 zeros, respectivamente. Em ambas as aplicações, a *CLA* novamente apresentou menor número de ciclos para a execução em relação à *CPU*. Esta redução foi de 1.23 vezes e 1.31 vezes nas respectivas aplicações.

Nos três experimentos, a *CLA* conseguiu apresentar resultados melhores em desempenho do que a *CPU*. Isto é devido à *CLA* poder acessar diretamente módulos de periféricos. Como a *CLA* fica responsável exclusivamente para a execução destas tarefas de controle, já era de se esperar que a mesma fosse ter uma performance melhor que a *CPU*.

7 Conclusões

Os quatro aceleradores de processamento (*FPU, TMU, VCU-II* e *CLA*) apresentam benefícios imprescindíveis para controles de processo em que são necessárias respostas de controle o mais rápido possível.

Com o advento de novas tecnologias e implementações na área da eletrônica nas últimas décadas, foi se tornando viável a inserção de tais aceleradores em microprocessadores, que vieram a receber o nome de *DSPs*, mantendo um tamanho razoável do dispositivo e proporcionando preços acessíveis até mesmo para aplicações de baixo custo. Como maneira de apresentar todos estes resultados de melhora no desempenho do processamento de sinais, a *Texas Instruments*™ apresentou diversos testes realizados com cada um destes aceleradores, e conseguiu mostrar tais resultados, abrindo as portas para os DSPs tanto na indústria quanto em pequenas aplicações.

Ter o entendimento dos conceitos e propriedades de cada um destes aceleradores faz com que o programador ao utilizar um *DSP* em sua aplicação, consiga obter a melhor eficiência do dispositivo. O estudo das referências bibliográficas e a escrita deste documento foi propícia para obter uma visão mais detalhada e profunda acerca dos aceleradores de processamento presentes nos *DSPs*, principalmente os contidos no TMS320F28377S.

Referências Bibliográficas

[1] SPRS881D – TMS320F2837xS *Delfino™ Microcontrollers Datasheet*. Ago. 2014, revisada em Jul. 2017.

Disponível em:

www.ti.com/lit/ds/symlink/tms320f28375s.pdf

Acessado em 07/02/2018 às 09h58.

[2] SPRUI25C – *LaunchXL – F28377S Overview, User's Guide.* Jun. 2015, revisada em Abr. 2017.

Disponível em:

www.ti.com/lit/ug/sprui25c/sprui25c.pdf

Acessado em 07/12/2017 às 15h13.

[3] Padrão IEEE-754.

Acesso em:

pt.wikipedia.org/wiki/IEEE_754

Acessado em 02/01/2018 às 18h16.

[4] SPRUHS1A – TMS320C28x *Extended Instruction Sets – Technical Reference Manual*. Mar. 2014, revisada em Dez. 2015.

Disponível em:

www.ti.com/lit/ug/spruhs1a/spruhs1a.pdf

Acessado em 04/01/2018 às 13h01.

[5] SPRAAN9A – *Application Report* – TMS320C28x *FPU Primer*. Jul. 2009.

Disponível em:

www.ti.com/lit/an/spraan9a/spraan9a.pdf

Acessado em 04/01/2018 às 13h28.

[6] SPRY288A – *Technical Brief – Accelerators: Enhancing the Capabilities of the C2000™ MCU Family.* Jul. 2015, revisada em Nov. 2016.

Disponível em:

www.ti.com/lit/an/spry288a/spry288a.pdf

Acessado em 04/01/2018 às 14h16.

[7] SPRU566L – *C2000™ Real-Time Control Peripherals – Reference Guide.* Jun. 2003, revisada em Jun. 2015.
Disponível em:
www.ti.com/lit/ug/spru566l/spru566l.pdf
Acessado em 05/01/2018 às 18h10.
[8] *Direct-Quadrature-Zero Transformation.*
Acesso em:
en.wikipedia.org/wiki/Direct-quadrature-zero_transformation
Acessado em 05/01/2018 às 19h25.
[9] Transformadas de Clarke e Park. LIMA, Kleber, UFRJ, Slides de aula.
Disponível em:
www.coe.ufrj.br/~kleber/Transformadas.pdf
Acessado em 05/01/2018 às 19h47.
[10] SPRUHX5E – TMS320F2837xS *Delfino™ Microcontrollers – Technical Reference Manual.* Ago. 2014, revisada em Set. 2017.
Disponível em:
www.ti.com/lit/ug/spruhx5e/spruhx5e.pdf
Acessado em 15/01/2018 às 18h12.